6-1-2000

Depth from Flash

David B. Martin
Dartmouth College

Depth from Flash

David B. Martin

Spring 2000

Abstract

Digital camera technology has recently seen substantial improvements in image quality while lower prices have made it affordable to the average consumer. Camera manufacturers, however, are not taking full advantage of this new medium for image capture. By filtering the already digitized image produced by these cameras through on-board image processing algorithms we can dramatically increase the power of digital cameras. For example, according to experts in the photographic industry, most people simply take bad pictures. Classic examples of this phenomenon are photographs taken indoors with a point-and-shoot style camera using its built-in flash. The subjects of these photographs often seem to have a spotlight on them, making them look bright and washed out while the rest of the photograph is dark and indistinct. This can primarily be accounted for by a well known property of point light sources: falloff in brightness is inversely proportional to the square of the distance between the light and the object being illuminated. A technique first introduced in the field of computer vision has been shown to successfully recover information about the distance between the light source and objects in the world [4]. We propose using this technique, which is readily implementable in hardware, to correct for a variety of poorly illuminated digital images.

1 Introduction

Photography has always been a sexy discipline. Embodying the perfect mix of artistry and technical mastery, the superior photographer is respected as a historian, a poet, and, since Madison County became famous, a lover. This makes it seem particularly unfortunate that taking good photographs is often as difficult as convincing a subject to pose for them. Indeed, it seems likely that the two are correlated.

Since their introduction to the consumer market, engineers have been searching for techniques to make the process of taking pictures more convenient and rewarding– the Polaroid camera is an example of one fantastic success. Shortening the development process only tackles one aspect of the problem consumers have with photography though. A more fundamental frustration with photography involves the truth that it is simply difficult to take good pictures. Anyone who has excitedly snapped a shot of an enormous natural vista or of a performer inside a darkened theater only to see the film develop to a disappointing blur has experience with this reality.

Coming up with a "silver bullet" solution to all photographically intractable situations would make one fabulously wealthy and famous. Trying to think of one, however, is time wasted. A myriad of factors combine to affect the final exposure in the photograph of even the simplest scene: ambient and artificial light sources, lens focus, exposure time, and, most importantly to the clumsy photographer, motion. Traditional cameras, which use silver-halide chemicals to record their exposures, do not provide any means for correcting any of these influences once the exposure has already been made.

Digital cameras are an entirely different story. It is already common practice, even within the most inexpensive consumer-grade digital cameras, to include custom hardware on-board the device for manipulating and improving the image after the exposure is complete. Building

smarter cameras is no longer a problem for exclusively the physicist– it is now very much the domain of computer science.

One of the most common problems that amateur photographers encounter is working with subjects under poor lighting conditions. Usually this means when the photographer wants to take a photograph indoors. Strobe flashes have greatly simplified the process, but have also made common some annoying artifacts. Most pictures taken with a flash have an over-exposed subject, while the background is dark and indistinct. If we consider the flash to be a point light source (as it is reasonable to do in most circumstances), we can account for this problem by recognizing that the brightness of a point light source diminishes according to the square of the distance between itself and the subject. Objects within a tight space of foreground (i.e., the subject) are thus well lit, while objects even a short distance beyond are darkened beyond easy recognition.

Using a technique that was first introduced in the field of robotic vision [4], we will develop a computationally inexpensive method for improving the results of images taken under poor lighting conditions.

To provide the reader with appropriate background, an overview of the camera obscura and the perspective projection model is presented in section 1.1 and 1.2. A short background of modern digital camera technology is presented in 1.3 to acquaint the reader with the most commonly used technologies in the field. Finally, the particulars of using a flash to take pictures are considered in section 1.4. Readers already acquainted with the field may wish to skip to section 2.

1.1 Camera Obscura

The simplest camera consists of an enclosed box with a photographic plate sitting on one side opposite a "pinhole", a infinitesimally small puncture in the face of the box to let light

pass inside. This type of camera is traditionally referred to as a camera obscura, and early photographic plates were produced in just this fashion [3].

Figure 1 illustrates the important features of a camera obscura. In the figure, we have some arbitrary real world object depicted as an irregular polygon. Some point $\vec{P} = (X, Y, Z)$ reflects a ray of light through the pinhole of the camera, here depicted at the origin (though any coordinate system could be chosen), and intersects the plane of the image sensor at point $\vec{p} = (x, y)$. The distance between the pinhole and the plane of the image sensor is referred to as the *focal length*, and is labelled d in the figure. As the figure depicts, the point $\vec{p}$ may exist as any (x, y) pair within the confines of the image sensor's plane.

With this device and terminology in mind, we can move on to discuss the mathematical underpinnings of this model, all of which surround two equations that define how a point on the plane of the image sensor is related to any arbitrary point in the world.

1.2 Perspective Projection

One of the simplest and most useful means for talking about image formation is the perspective projection model. Image formation can be regarded as the process of projecting some object in R^3 to the plane of an image sensor in R^2. For convenience, in figure 1, the origin is depicted as coinciding with the pinhole of the image sensor.

Conversion between world and image coordinate systems can be accomplished by observing that the focal length (d in figure 1) and a point on the image sensor form similar triangles with the distance Z and the point in the world corresponding to the chosen location on the image sensor. The equations for this relationship are called the perspective projection equations:

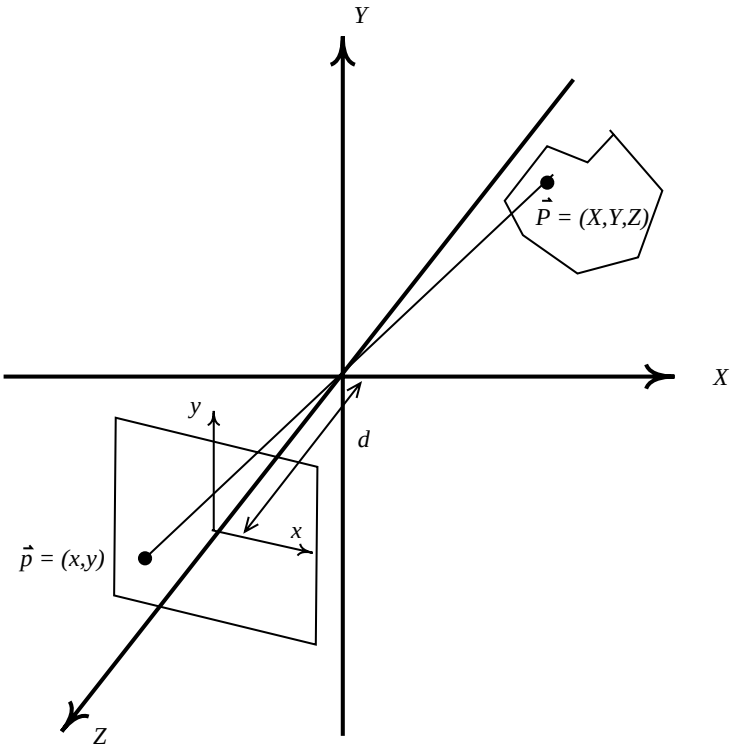

Figure 1: Using the perspective projection model, rays of light travel from the surface of an object in world coordinate space, pass through the pinhole of the image capture device, and are recorded on the image sensor. Within a specific range limited by the size of the image sensor and the focal length of the device, each point in world coordinates corresponds to exactly one point in image space. The relationship between these two points in governed by the perspective projection equations (equation 1).

$$x = -\frac{dX}{Z} \qquad \text{and} \qquad y = -\frac{dY}{Z}. \tag{1}$$

In the operation of any camera, there is an inherent loss of information as an image is created. With the perspective projection model, all objects along an identical ray are projected to the same image coordinate. In so doing, information about depth is irrevocably lost.

1.3 Digital Cameras

As quickly as digital cameras have developed over the past several decades, there are a few characteristics that both the earliest and the most modern cameras share. Digital images generally consist of a two dimensional array of grayscale pixels. Depending on the quality of the device, the size of these pixel arrays can vary from that of the earliest devices (around 320x240 pixels), to today's cameras, which can produce arrays of 1600x1200 pixels or more. It has become popular to refer to the number of pixels in these arrays in units of megapixels. A 1600x1200 pixel array has 1.9 million pixels, or 1.9 megapixels.

While in recent years CMOS technology has grown in popularity due to its relatively inexpensive manufacturing process, the vast majority of these digital imaging devices rely on a technology called CCD, or Charge Coupling Device. CCD sensors work by measuring the amount of light that comes into contact with their surface, and storing an electric charge proportional to the intensity of their exposure. Thus, a CCD sensor exposed to bright, direct sunlight will store a substantial charge relative to a neighboring sensor that is only "seeing" shade. These charges are then fed to an analog-to-digital converter that produces values for the intensity of light at each pixel [7].

If one treats these values (which are specified in binary) as unsigned integers, one can measure the sensitivity of a camera by the arity of its pixel values. The latest cameras can express twelve whole bits of information about each pixel, but eight bit devices are still the most common. With eight bits per pixel, a grayscale image has 255 different values to work with between pure black and pure white; a limitation that can produce noticeable artifacts of quantification within the captured images.

Two different methods are popularly applied for reproducing color in digital images. In the first method, red, green, and blue color filters are placed over individual CCD sensors in specific patterns over the entire face of the array. By combining the sampled intensities of

several neighboring sensors, an approximation can be made for the color of the light reaching a point in the image. Three totally separate arrays of CCD sensors are used in the second method. Light entering the camera is split optically into three different channels. Each identical stream of light is made to expose its own CCD array; one filtered for red, one for blue, and one for green. This method yields better results than pattern-based color recovery, but manufacturing the devices is significantly more complicated. Some professional digital cameras use a third method where one CCD sensor array is exposed and filtered for each color in order, but the time needed for each exposure makes this technique impractical for uses much beyond still life type photographs.

Unfortunately, the CCD sensors are subject to electro-mechanical failures; misreporting or mis-measuring the amount of light reaching the sensor. If a sensor makes a mistake by building too much charge or too little charge for the amount of light it was subjected too, the pixel will be too dark or too bright at that particular location in the image. An image plagued by many such pixels is said to be "noisy", and most digital cameras, especially ones using CMOS sensors, which can be particularly noisy when compared with CCD devices, use some sort of digital signal processor (DSP) to reduce noise.

Despite the engineering problems associated with collecting an image digitally, having an image in a digital format provides a plethora of advantages for the photographer. In most cases, the image is available more or less instantaneously and can be processed by any one of a number of other custom integrated circuits (ICs) that might be located on the camera itself. Several of the most affordable digital cameras offer a digital zoom feature that isolates a particular portion of the raw image and doubles or quadruples the number of pixels drawn for each pixel actually recorded by the cameras sensors. The end result is an image that appears "zoomed in" two or four times, though the loss of image resolution is readily apparent.

Techniques like this one, and the noise-reduction algorithms already mentioned, provide a good first step in exploring the capabilities and advantages of digital cameras. Many more advantages remain untapped, and we hope to provide the digital imaging world with an important new technique for improving images.

1.4 Flash Photography

While digital cameras are wonderful, it is still extremely difficult for the average amateur photographer to take high quality images, particularly in poor lighting circumstances. Digital technology can help solve some of these problems by applying new image processing techniques.

A particularly perplexing problem is presented by indoor photography. Without the aid of sunlight, photographers generally find their subjects to be poorly lit, and thus must rely on some artificial means of illumination. One method of artificially lighting a subject, familiar to the amateur and professional photographer alike, is the use of a flash.

Historically, magnesium powder charges were ignited inside of burnished metal tins to provide a blinding flash of artificial light. Besides the physical danger of using these flammable charges, it was difficult to produce successive flashes quickly. With the introduction of strobe flash technology, indoor photography became a great deal safer, and quickly increased in popularity. The widely acclaimed Polaroid camera prominently included a large flash bulb, further fueling amateur indoor photography (and subtly hinting at the future success of digital "instant" cameras).

When speaking mathematically of a flash as a light source, it is modeled as as a *point light source*. Rays of light leaving the light source can be regarded as having left some ideal point. This stands in particular contrast to an *ambient light source* for which no location may be discerned as the source of light.

Figure 2: James Stewart made good use of his camera's flash in Alfred Hitchcock's classic Rear Window

One of the natural properties of a point light source is that its brightness decreases quadratically. As the distance between a light and the object it illuminates increases, the brightness decreases by the square of the distance between the two objects. For the photographer, this means that using a flash to take pictures of a wide range of subjects can be tricky. In particular, objects in the foreground are often lit to brightly, and look "washed out", while objects in the background are poorly lit and indistinct. Other problems arise while trying to take flash photographs of shiny or high gloss surfaces like glass or water. Perhaps the most common problem with flash photography is the "red eye" effect. Light bouncing off the blood vessels of the human retina can give an eerie blood red color to the pupil of the subject's eye; a problem that several consumer digital and analog camera manufacturers have been vying to solve in recent years.

In order to continue our discussion of flash photography, we must introduce a few concepts that will be useful in order to rigorously define our notions of digital images lit by a flash.

In figure 3 we can see how the brightness at each point in the image can be expressed as a function of the light reaching the surface of each point in the world. No ambient light sources are depicted in this figure. This relationship is formalized in equation 2:

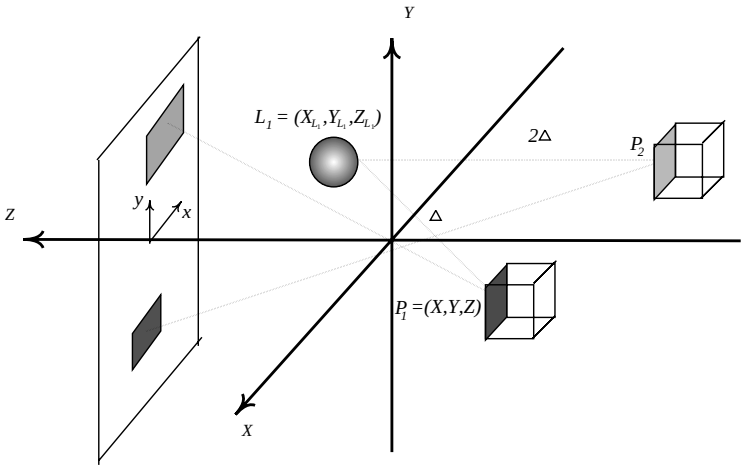

Figure 3: The brightness at an arbitrary location (x, y) in the local image coordinates is a function of the light reaching the surface of a location in world coordinates (X, Y, Z). The farther light travels from the point source, the dimmer the corresponding location in image coordinate space.

$$I(x, y) = \frac{L \cdot R(X, Y, Z)}{\Delta^2(X, Y, Z)}, \tag{2}$$

where I is the function of brightness as defined at each point (x, y) of the image, L is a scalar defining the brightness of the flash, R is the reflectance function of the subject at each point (X, Y, Z) in world coordinates, and Δ is the function of absolute distance between the light and each point (X, Y, Z) in the world, as given by:

$$\Delta(X, Y, Z) = \sqrt{(X_L - X)^2 + (Y_L - Y)^2 + (Z_L - Z)^2}, \tag{3}$$

where the light coordinates, (X_L, Y_L, Z_L) represent the location of the light source.

As illustrated in figure 3, the quadratic falloff is modulated by the distance between light source and object. If we use the values for distance depicted in the figure, we arrive at the following two equations for world coordinates P_1 and P_2:

Figure 4: *Left*: Quadratic falloff in brightness leaves the subject with an unfortunate halo, while the rest of the car is difficult to see. *Right*: All areas of the photograph are easy to see, including elements of the background.

$$I_1 = \frac{L \cdot R}{\Delta^2(X_1, Y_1, Z_1)} \qquad \text{and} \qquad I_2 = \frac{L \cdot R}{4\Delta^2(X_2, Y_2, Z_2)}. \tag{4}$$

Here, Δ and 2Δ represent the distance between the object and light sources one and two respectively, as calculated with equation 3. Thus, we observe that with twice the distance to P_2, four times less light reaches the object's surface.

This creates a host of problems for the real world photographer. For example, at left in figure 4 is a poorly taken photograph. The subject of the photograph seems to be in a spotlight, while other areas are indistinguishable. The photograph at right is well illuminated. Objects in the background are readily discernible, and while the eye is invited towards the subject, other areas of the photograph can be considered with equal clarity.

In real world situations, it is reasonable to think that we might have some knowledge of L, the brightness of our flash, at hand. If we somehow knew the value Δ or the function R, we could solve to recover the remaining unknown and the image could be readily corrected. By scaling the intensity of the pixel for the estimated distance to its real world object, the

entire image could be returned to an "ideal" illumination. With depth information we could render a new image with a light source in a completely different location. Or we could render the image with several light sources. Only imagination and compute cycles limit what we can do with the image given a depth map.

Unfortunately, in moving from R^3 to R^2, we introduced an inherent ambiguity in depth, which is not recoverable from this single image. In the next section, we will explore the nature of this ambiguity in more detail, and provide a method for overcoming it. Using this method, we will finally be able to correct our images' problems with lighting.

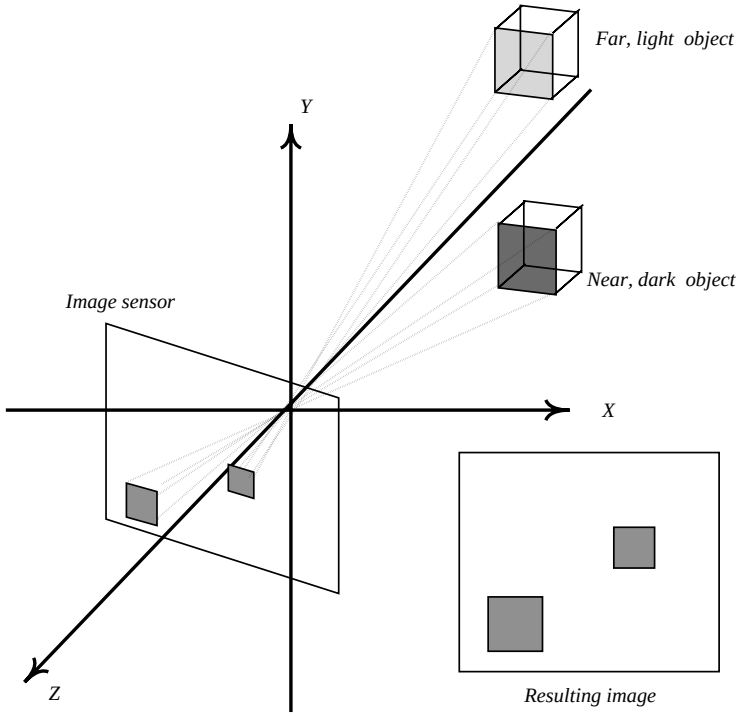

Figure 5: If two cubes in the world have appropriate reflectance functions, then the amount of light reaching the image sensor will appear to be the same, and we will be unable to distinguish from the image alone whether both cubes are equidistant-distant from the sensor, or if one is further than the other.

2 Depth From Flash

A good thought experiment is in order to help us arrive at a means for exacting this measure of distance; a measure we just discovered is the key to achieving our goal of correcting poorly illuminated images taken with a flash.

Imagine taking a picture of two different cubes, each at a different distance from the camera in a darkened room. Suppose that one cube is light gray, while one cube– the nearer one– is darker in appearance. If the room is completely dark when we take the flash photograph, the cubes are placed at the right locations, and the reflectance function of each cube is chosen correctly, we we will notice something very interesting about the resulting image. Let us choose a set of such conditions and calculate values for intensity in the image

for the points at which each cube is projected.

Let cube one be a distance Δ_1 from the point light source with reflectance function R_1. An easy shorthand for expressing the intensity of each pixel in the image for this cube is:

$$I_1 = \frac{L \cdot R_1}{(\Delta_1)^2} \tag{5}$$

where R_1 and Δ are actually functions over world coordinates (X, Y, Z) as defined in equation 3 above, and I is a function over the image coordinates (x, y). If we define cube two as having reflectance function R_2 such that:

$$R_2 = 4R_1 \tag{6}$$

and if we also define Δ_2 as:

$$\Delta_2 = 2\Delta_1 \tag{7}$$

then the intensity of pixels in the image cube two will be given by:

$$I_2 = \frac{L \cdot R_2}{(\Delta_2)^2} = \frac{L \cdot 4R_1}{(2\Delta_1)^2} = \frac{L \cdot 4R_1}{4(\Delta_1)^2} = \frac{L \cdot R_1}{(\Delta_1)^2}, \tag{8}$$

and I_2 would be equal to I_1 as shown in equation 5. The resulting picture would thus shows two identically shaded cubes.

On reflection, this agrees with our intuition for the situation. Although the closest cube is dark, the flash lights it well. The furthest cube reflects lots of light, but not very much light reaches it. Thus, the final image shows two cubes with the same brightness, and leaves us with an ambiguity in distance from the camera.

When we consider the full implications of this situation, we realize that there is no way to resolve the matter with the information at hand. With only a single photograph, we have no facility for learning anything further about the world beyond the camera's lens.

But what if we were working with a second piece of information? One idea that may leap to mind is to use a second camera to photograph the scene. With a second image from a slightly different perspective, we would gain information about depth in the world. The reader should be quick to recognize that what we are talking about is stereoscopy [1].

Stereoscopy is, in brief, a technique for determining distance from a pair of spatially offset pictures. If we know the amount the image sensor was offset to produce the pictures, we can compare the location of identical objects in the two images and come up with an estimate for their distance from the sensor. Close objects will move a large amount relative to objects that are far away.

There are, however, some pragmatic problems with this technique. How practical is it to have two cameras simultaneously snap pictures of the same scene? We might be able to design such a device, but CCD manufacture is an expensive process, and building a camera with two arrays of sensors would be too costly to be economical for the mainstream market. Moreover, stereoscopy relies on being able to identify the same location on objects in the world across two pictures. Recognizing the same features in two different images is an entire image processing problem unto itself [3]. Given the hard problems presented by stereoscopy, can we come up with a more effective solution?

What if we used the same camera to take two pictures, but moved the flash by some arbitrary amount when we snap the second picture? Assuming that neither the camera nor the world moves between shots, we would have two pictures that vary only in brightness.

By sliding the flash a little closer, for example, to the subject in the second picture, the entire image would appear a little brighter, but in some very specific ways. Returning to our thought experiment, we would see the dark cube increase in brightness a lot, while the white cube increased by very little. A careful comparison of these two pictures with thought to the quadratic falloff in the flash's brightness would then give us our desired clues about

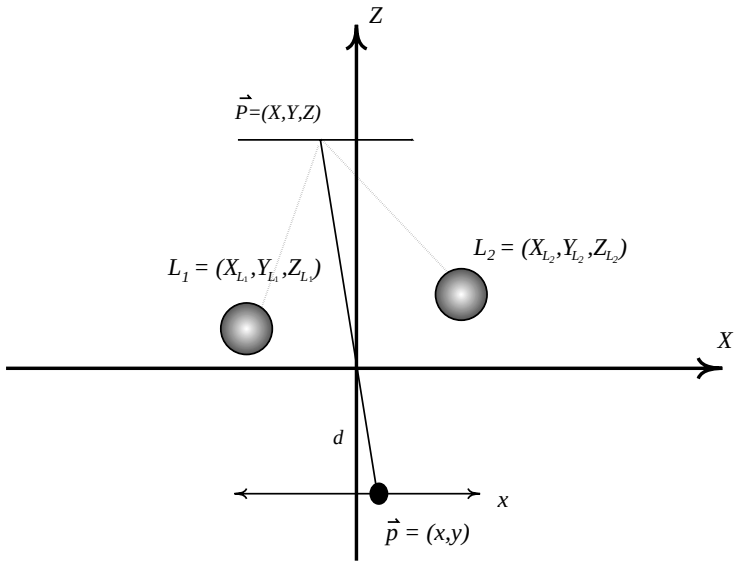

Figure 6: Light reaching a point in the image at point (x, y) will vary depending on the position of the light source with respect to the object.

distance. It would suddenly be evident that the lighter cube must be the most distant, and the darker cube the closest!

Conversely, by sliding the flash a little further from the posters, the black poster would drastically decrease in brightness, but there would be little change in the color of the white poster. This would give the same clues about the relative locations of the posters as when the flash was moved closer. Further reflection affirms that the same is true even when moving the flash to the right or the left. In the next section we will outline this technique for distance recovery using flash in full detail.

3 Methods

In the world coordinate system, two lights are located at arbitrary positions, as defined by the points L_1 and L_2 respectively. Recall from the previous section that the functional shorthand for defining the intensity of light for each point in the image is given by:

$$I = \frac{L \cdot R}{(\Delta)^2}, \tag{9}$$

where L is the intensity of the light source, R is the reflectance of the object in the world, and Δ is the distance from the light source to the object. In figure 6 this would be the distance between the light source– either L_1 or L_2. The reflectance function would be that of the object at point $\vec{P}$.

Equations 10 and 11 below represent the intensity functions for light reaching the image sensor from two different light sources with arbitrary positions:

$$I_1 = \frac{L \cdot R}{(X_{L_1} - X)^2 + (Y_{L_1} - Y)^2 + (Z_{L_1} - Z)^2} \tag{10}$$

$$I_2 = \frac{L \cdot R}{(X_{L_2} - X)^2 + (Y_{L_2} + Y)^2 + (Z_{L_2} + Z)^2}, \tag{11}$$

where L and R are as before, and the 3-tuples $(X_{L_1}, Y_{L_1}, Z_{L_1})$ and $(X_{L_2}, Y_{L_2}, Z_{L_2})$ represent the position of the light sources in world coordinates.

We must remove the $L \cdot R$ term of these equations if we are to make any progress towards isolating the depth Z. Consider taking the ratio I:

$$\text{Let} \qquad I = \frac{I_1}{I_2} = \frac{(X_{L_2} - X)^2 + (Y_{L_2} - Y)^2 + (Z_{L_2} + Z)^2}{(X_{L_1} - X)^2 + (Y_{L_1} - Y)^2 + (Z_{L_1} - Z)^2}. \tag{12}$$

Note that the $L \cdot R$ term from both equations cancel, as desired.

Let's step back to regain our bearings then. With what exactly does equation 12 provide us? We know the value of the left hand side– some constant term for intensity at a particular point in the image. We also assume knowledge of the coordinate positions of the two lights. That is, the 3-tuples $(X_{L_1}, Y_{L_1}, Z_{L_1})$ and $(X_{L_2}, Y_{L_2}, Z_{L_2})$ can also be regarded as known constants.

We are, however, out of arguments when it comes to the world coordinates of the object. With three unknowns, X, Y and Z there doesn't seem to be much hope for calculating any new piece of information. Can we do anything to rid ourselves of some or all of these terms?

Let us consider replacing the X and Y terms with their values from the perspective projection equations:

$$X = \frac{Zx}{d} \qquad \text{and} \qquad Y = \frac{Zy}{d}, \tag{13}$$

produces the following equation:

$$\text{Let} \qquad I = \frac{I_1}{I_2} = \frac{(X_{L_2} - \frac{Zx}{d})^2 + (Y_{L_2} - \frac{Zy}{d})^2 + (Z_{L_2} + Z)^2}{(X_{L_1} - \frac{Zx}{d})^2 + (Y_{L_1} - \frac{Zy}{d})^2 + (Z_{L_1} - Z)^2}. \tag{14}$$

Is this equation any better than equation 12? Definitely. While the world coordinates X and Y are as lost to us as their friend Z, the image coordinates x and y are decidedly not. As we noted above, I is actually a function of intensity over the two dimensional array of pixels. Thus, x and y can rightly be considered known constants as easily as I can be treated as such.

Suddenly, we find ourselves with one equation in one unknown. Moreover, we have an equation with one quadratic term, one linear term, and one constant term; perfect for plugging into the binomial equation. Cross multiplying:

$$I\left[\left(X_{L_1} - \frac{Zx}{d}\right)^2 + \left(Y_{L_1} - \frac{Zy}{d}\right)^2 + (Z_{L_1} - Z)^2\right] = \left(X_{L_2} - \frac{Zx}{d}\right)^2 + \left(Y_{L_2} - \frac{Zy}{d}\right)^2 + (Z_{L_2} - Z)^2 \tag{15}$$

$$I \cdot \left\{\left[X_{L_1}^2 - 2X_{L_1}\frac{Zx}{d} + \frac{Z^2x^2}{d^2}\right] + \left[Y_{L_1}^2 - 2Y_{L_1}\frac{Zy}{d} + \frac{Z^2y^2}{d^2}\right] + \left[Z_{L_1}^2 - 2Z_{L_1}Z + Z^2\right]\right\} -$$

$$-\left[X_{L_2}^2 - 2X_{L_2}\frac{Zx}{d} + \frac{Z^2x^2}{d^2}\right] + \left[Y_{L_2}^2 - 2Y_{L_2}\frac{Zy}{d} + \frac{Z^2y^2}{d^2}\right] + \left[Z_{L_2}^2 - 2Z_{L_2}Z + Z^2\right] = 0 \quad (16)$$

Organizing into quadratic, linear, and constant terms respectively yields:

$$\begin{aligned} Z^2\left\{\left(\frac{I\cdot x^2}{d^2} + \frac{I\cdot y^2}{d^2} + I\right) - \left(\frac{x^2}{d^2} + \frac{y^2}{d^2} + 1\right)\right\} + \\ Z\left\{\left(\frac{-2I\cdot X_{L_1}x}{d} - \frac{2I\cdot Y_{L_1}y}{d} - 2I\cdot Z_{L_1}\right) - \left(\frac{-2X_{L_2}x}{d} - \frac{2Y_{L_2}y}{d} - 2Z_{L_2}\right)\right\} + \\ + \left(I\cdot X_{L_1}^2 + I\cdot Y_{L_1}^2 + I\cdot Z_{L_1}^2\right) - \left(X_{L_2}^2 + Y_{L_2}^2 + Z_{L_2}^2\right) \end{aligned} \quad (17)$$

which are ready to be plugged into the binomial equation:

$$Z = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (18)$$

where a, b, and c are the coefficients of the quadratic, linear and constant terms of equation 17 defined above.

Here we pause to recognize just what we have calculated. This term Z can be calculated for every point in the image. That is to say, we have just generated an estimate for depth between camera and the objects in the world at every point in the world the image was able to capture on its sensor. We have a depth map.

3.1 Simulations

Using MATLAB, a set of scripts were created to implement the polynomial approximator, and are included in Appendix 1. For the initial set of tests, the camera was set facing down the z-axis, and a cube was placed a finite distance 20 units ahead of it, filling the camera's field of view. Figure 7 includes the raw images generated by MATLAB. Local averaging was performed using a weighted vector (1 4 6 4 1). These images used sixteen bits of quantification, and included no noise or ambient light.

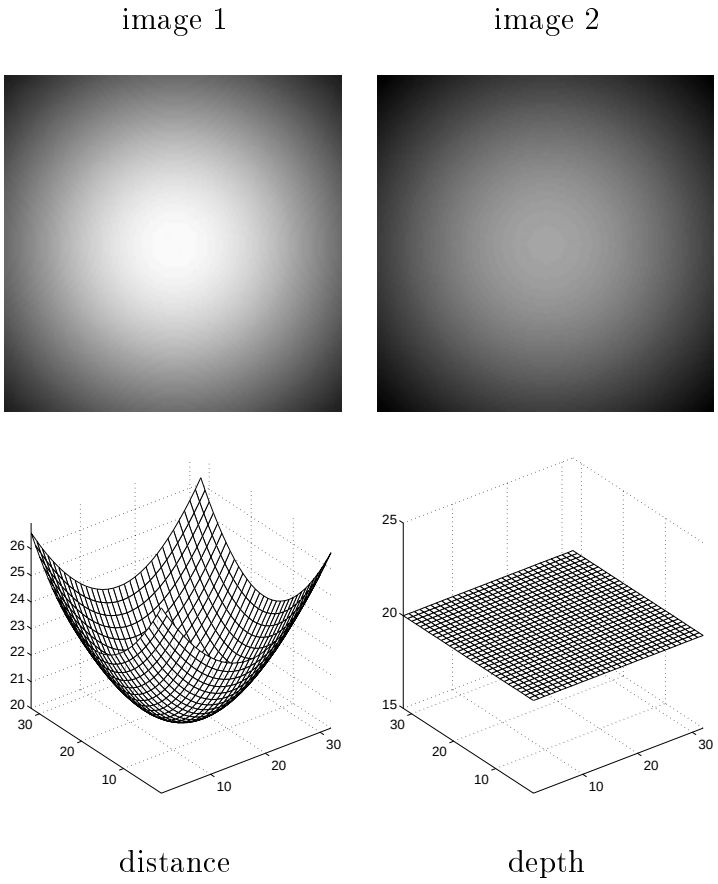

Figure 7: *Top row*: MATLAB-produced raw images of a cube. The cube's face sits parallel to the plane of the image sensor 20 units away (an arbitrary choice of value). 10 bits per pixel were used to render the grayscale in each image. *Bottom row*: Distance estimate produced by the polynomial approximator using images from the top row. The estimate is shown from different views to aid in interpreting the results.

The result of running the polynomial approximator on the two inputs is also depicted in figure 7. The depth map in particular gives good encouragement as to the accuracy of our technique– it shows the face of the cube smoothly at 20. The distance map also produces the parabolic surface we expected. Points at the corners of the plane are the most distant, while the center of the cube is the closest.

3.2 Sensitivity

To gain a better understanding of how the approximator functions under different conditions, a series of new experiments were constructed. As was the case with the data from figure 7, in all of these tests the camera was kept 20 units from the model. The position of the lights with respect to the camera was varied by experiment, but unless otherwise specified, the arity for each pixel in the generated raw images was 10 bits.

In all our initial models, the world is assumed to consist of a plane parallel to the plane of the image sensor, and translated by a specific distance ahead of the sensor on the z axis. Each of the figures in this section were generated using 20 units as the ground truth value for the sensor-object plane distance. The $L \cdot R$ value used to calculate the ground truth images was 100, and the focal length was set to 0.5.

Intuition tells us that with light sources very close together, we may not be able to produce good estimates. Just as a picture taken with low light will produce a poor exposure, so two images with the light source in virtually the same location will produce a poor ratio (i.e., very close to one). Thus, our first instinct besides checking that the technique works at all is to test how the positions of the light sources– relative to each other and relative to the camera– affect the error of the depth estimate.

Figure 8 presents the error in our depth estimate, calculated using the root mean squared method, as a function of the distance between the camera and the lights.

Although the error is not linear as distance increases, we do not find exponential growth in error either. As a whole, the error is quite small– less than 1% across most of the offsets.

How well does the polynomial approximator function when we increase the delta between the two light sources? In figure 9 we held the camera and one light source stationary, while steadily increasing the distance of the second light source from the first.

As our intuition indicated, the larger the distance between the two light sources, the

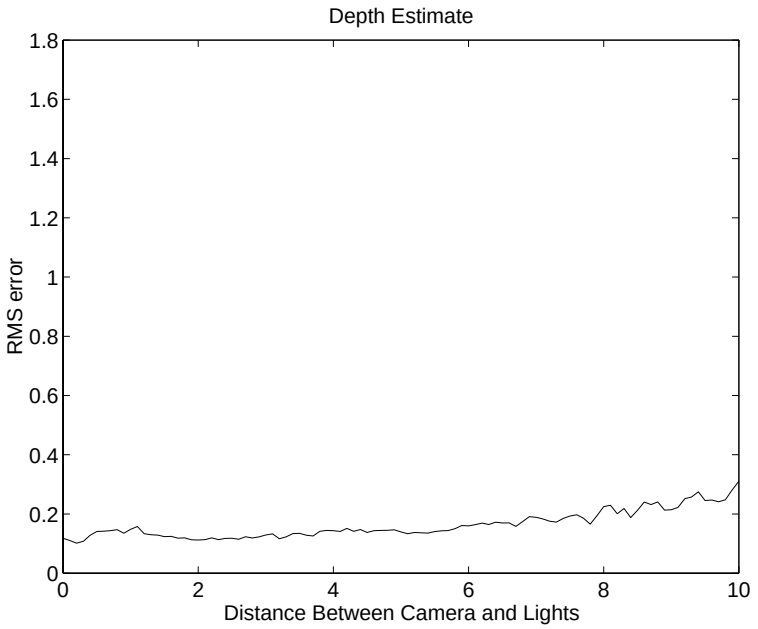

Figure 8: Error in estimation of depth as a function of the distance between camera and lights, where lights are kept stationary with respect to each other. The surface of the object was at a distance 20 units from the origin.

better our depth estimate becomes. Fortunately, very little distance is needed in order to achieve excellent error tolerances. Error is already about 1% when the lights are moved apart by 1/20th the distance between camera and object in world.

To put this in perspective (no pun intended, of course), we note that almost all consumer cameras use a strobe flash with a range of about 20 feet. Assuming we take a picture of a subject at the edge of this range, we would only have to move the flash about a foot between exposures to achieve about 1% error in a calculation of depth from the two pictures.

In the real world, a variety of factors beside the photographer's light source, the actual object, and the camera affect the outcome of an exposure. As we noted in section 1, digital cameras have electro-mechanical sensors that are subject to certain discrete failures during normal operation. Specifically, sensors may inadvertently report values for intensity that are too high or too low given the actual amount of light that they received. In a digital image, noise appears as tiny specks in the image, and can give a grainy appearance to the otherwise

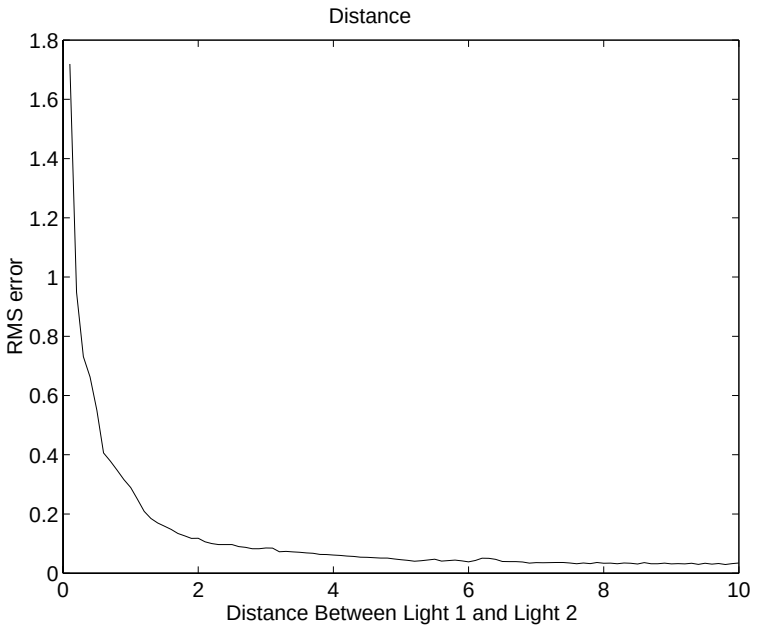

Figure 9: Camera and one light stationary; horizontal axis measures distance between position of first and second light.

sharp photograph. Apart from the avant-garde photographer, noise is not desirable in digital images, and manufacturers have gone to great lengths to produce cameras that keep noise in their images at a minimum.

Noise presents a special problem for the polynomial approximator. Given that a value for intensity is incorrect, our entire estimate for depth will be off. Figure 10 shows the error in approximation as a function of the noise in an image (measured in decibels). For this experiment, white noise was added to the image.

Clearly, even extremely low signal-to-noise ratios can produce reasonable depth maps for any given set of images, with errors of less than 1% being achievable around signal-to-noise ratio (SNR) values of 15. Again, we find the lovely non-linear drop in error with increase in SNR.

While indoor photographers often seek it out, ambient light has a detrimental effect on the polynomial approximator for the same reason that noise does. Unlike the case of noise,

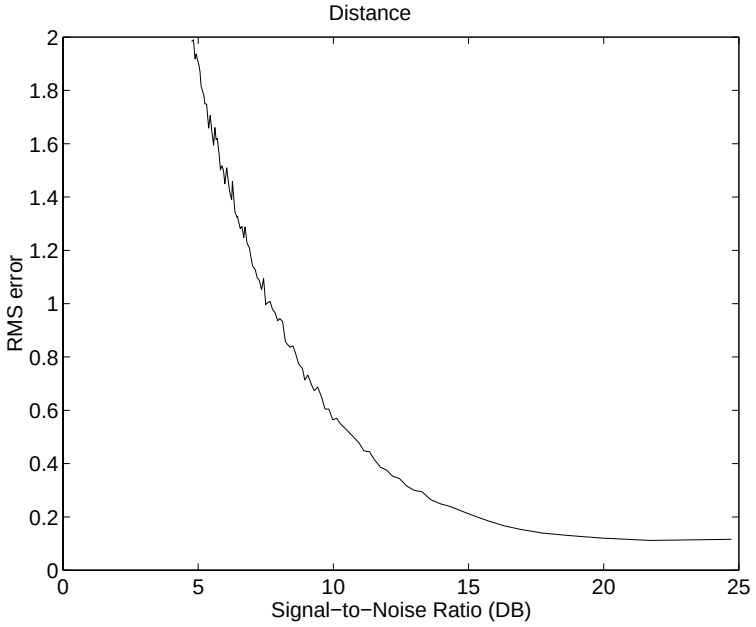

Figure 10: Error as a function of noise

however, an ambient term offsets all pixel estimates by some fixed amount. This is because equation 9 becomes:

$$I = \frac{L \cdot R}{\Delta^2} + A \tag{19}$$

where A represents some constant term of ambient light.

This presents us with a difficulty since we cannot cancel out this additive term using a ratio. How much an effect does this additive term have?

Figure 11 shows how the introduction of ambient light affects the depth estimates. Here we are happy to see a linear trend in the data. RMS error, however, is no longer in the satisfying range that we found it previously.

To cancel out this ambient term, one technique in particular suggests itself. By taking a third image, without any of our light sources illuminating the scene, we could obtain a precise value for the ambient term A at each pixel of the image. By subtracting this value A from

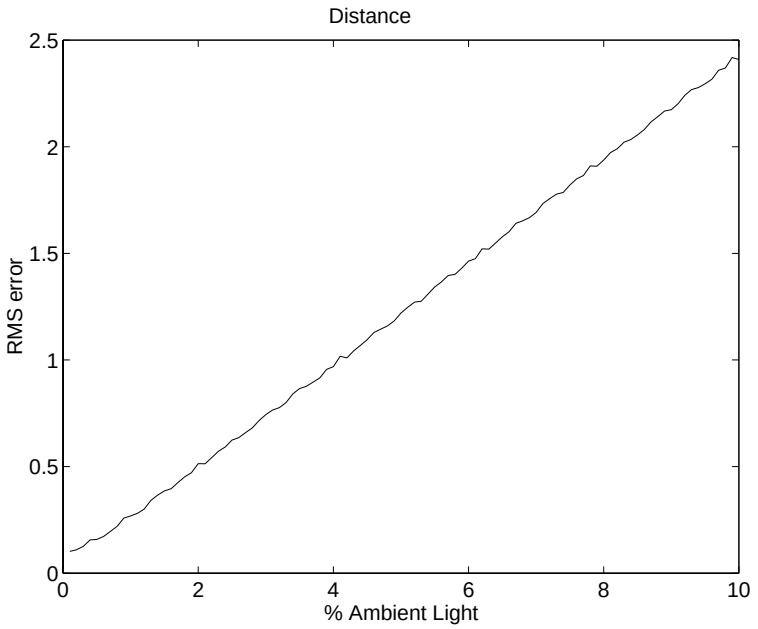

Figure 11: Error as a function of ambient light

the intensity of each pixel in the illuminated images, polynomial distance approximation could continue as normal; it would be as if the scene had been completely dark when the images were taken (i.e., ideal conditions for the polynomial approximator).

In practice, however, this technique may not be convenient to implement. Taking two separate images of a subject could already be a liability for the photographer interested in capturing a fast-moving subject; introducing a third only exacerbates the problem.

Alternatives to this technique might be a separate on-board mechanism photo meter for providing approximate data about ambient light at the time of an exposure. In most situations, ambient light is fairly diffuse, so this sort of "shotgun" approach may well be effective.

The number of bits used to approximate the intensity of light for each pixel (arity) has an obvious and direct effect on the quality of the polynomial approximator's depth estimates. To determine just how much of an impact bit arity has on the approximator, a fifth experiment

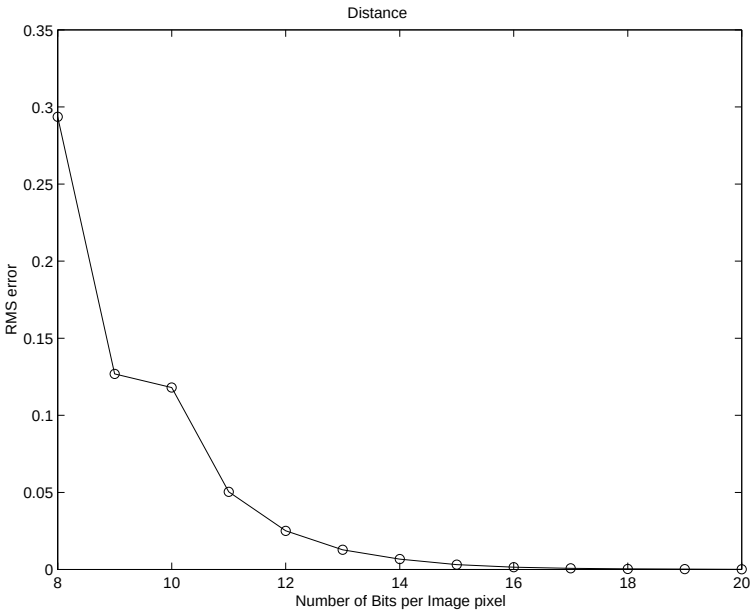

Figure 12: Error as a function of bit arity for pixels in image sensor

was run which left the lighting, camera, and object positions stationary while varying the arity of bits used to record each image. Results are shown in figure 12.

As expected, there is a sharply non-linear decrease in error with increases in bit arity [1]. Once the images are about 12 bits per pixel, there are virtually no gains in the quality of distance estimation.

3.3 The OpenGL Model

To continue our experiments, we needed to build a more flexible, substantial system for modeling real world subjects. A useful virtual model needed to support placing arbitrary subjects in R^3. In addition, we needed some way of rendering a quadratically attenuated light on the subject. The light, like the model, would have to be easily translated about

[1] Number of bits used to express the value for each pixel (e.g., 8 bit arity = $2^8 = 255$ different grayscale values.

the coordinate space. Finally, we would need some manner of recording the resulting scene for analysis. By building a virtual scene, and recording two images of the scene with the light in different places each time, we could successfully model a problem situation where the "double flash" technique is useful.

Fortunately, we didn't need to tackle the problem starting from scratch. OpenGL, a de facto industry standard application programming interface (API) for three dimensional graphics, offered all the features we were looking for in its rendering engine. The next section will provide a brief overview of the parts of the OpenGL rendering pipeline most important to our work. In the second section, we will describe in detail the program that was written to model our problem and collect data: Socrates. The final section will present several of the experiments.

3.3.1 The OpenGL API

The OpenGL Programming Guide introduces OpenGL as "a software interface to graphics hardware. (The GL stands for Graphics Library.) It allows you to create interactive programs that produce color images of moving three-dimensional objects." [8] As such, OpenGL is not a tool perfectly geared towards the application we had in mind. To begin with, only grayscale was desired. To create grayscale images, we saved OpenGL's screen buffers into TIFF images and converted the TIFF images to grayscale from their 24-bit RGB color content. While not the most desirable method, this was at least felicitous in the respect that it is analogous to the reverse function of an actual camera device.

In addition, because OpenGL is designed as a real-time rendering engine, and not a tool for producing accurate, high-quality stills of three-dimensional scenes, obtaining useful results from our models was not always easy. For example, to obtain an accurate number by which to perform pixel-to-world coordinate conversion, we were forced to coax numbers for

world-coordinate location on the x-axis out of the projection matrix (using gluUnProject).

While the lighting equations within OpenGL were accurate, achieving what intuitive behavior from the lights was also difficult. Moving the light more than 3 units from a unit cube left the cube in total darkness. We ran a series of experiments to determine the "sweet spot" along the z-axis which produced the most accurate quadratic attenuation. Moving the light too close saturated the object, but moving too far left the rendering engine without enough grayscale values to continue to express quadratic attenuation.

Ray tracing programs solved many of the problems mentioned, but lacked facilities for building interesting three-dimensional models quickly. In addition, there was the problem of significant overhead in rendering time for most ray tracers. With OpenGL, we were able to rapidly build and test a variety of different models using our own data-collection program, which is considered in the next section.

3.3.2 Socrates

Socrates was written using the C programming language. In addition to the standard C libraries, we used the OpenGL Utility Toolkit (GLUT) to facilitate writing the window-manager related portions of the code. GLUT took care of creating and destroying the X server window, as well as seeing that the appropriate function call backs were made upon various user interface events like keyboard clicks, mouse button clicks, and window resizing. GLUT is an open source library available courtesy of Mark J. Kilgard [6].

To record the images, we used the open source "libtiff", which provides a set of C functions for producing TIFF images. By marrying the appropriate OpenGL calls to libtiff, we were able to dump OpenGL's video buffer into a TIFF image file. [2]

[2] As a future warning to those who follow in our footsteps: it is prudent to pay careful attention to the number of buffers OpenGL is using to render its scene and the number of bits with which the X server is set

Socrates is built to read from a "scene" file at program initialization time. The scene file can contain an arbitrary number of lines, where each line represents a separate geometric shape in the final model. Cubes, spheres, cones, tori, dodecahedrons, octahedrons, tetrahedrons, icosahedrons, and teapots may all be specified in the scene fall.[3]

The size of each shape is specified on the line with the shape itself, as well as a pair of 3-tuples. The first 3-tuple specifies the location of the shape in three dimensional coordinate space. The second 3-tuple specifies the rotation about the x, y, and z axes respectively. Both 3-tuples may contain negative numbers.

By default, the camera sits at (0.0, 0.0, 3.0) and faces down the negative z axis (i.e., toward the origin). Once the scene is drawn, a variety of keyboard controls can be used to manipulate the model. At any time, the user can end the program by hitting the escape key.

The single light in the scene may be translated along the x axis using the left and right arrow keys, translation along the y axis is accomplished using the up and down arrow keys, and translation about the z axis is done with the page up and page down keys. The camera itself may be moved using the '[' and ']' keys.

By default, the light translates by 0.05 units with each key press. To increase or decrease this interval, the user can use the home and end keys.

Once the camera and light are in a suitable position, a variety of function keys are available for manipulating other aspects of the virtual world. To restore the scene to its initial state, the user can hit F1 at any time. An amusing feature that is not specifically useful to the program is the ability to rotate about the model. By pressing F2, the user can toggle this rotation on and off.

---

to display colors. Using any less than 24-bits will not produce desirable results, and the OpenGL interface will fail "silently" (it will happily render with only 16-bits without telling you anything about it).

[3]These geometric shapes are all available from the GLUT library, save the cube, which is hand tessellated for our model

If the scene includes a cube, sphere, cone, or torus shape, the tessellation of these shapes can be increased by using the F5 key. With each key press, the number of vertices used to render each shapes' surface is doubled. Thus, the number of surface normals is also doubled, and the realism of the lighting effect can be controlled within certain limits. Other shapes are not affected by increasing and decreasing the tessellation. To halve the tessellation, the user can press F6.

Keeping track of all the various functions of Socrates can be difficult, so by default some informational text is displayed across the top area of the model window. When an image is ready to be recorded, this information must be turned off, and this can be accomplished with the F9 key. F10 actually snaps a TIFF image of the scene exactly as displayed. A keyboard bell is sounded to indicate that the image has finished being recorded.

Positioning the light can be a tricky proposition if one has no indication of where the light is besides its effect on the model. To ease the manipulation of the light, a blue cursor appears by default around the position of the light within the OpenGL world. Since ambient terms of a material do not contribute to the main lighting equation, the cursor may safely be left on without affecting the rendering of the scene model. In most cases, however, it will likely obscure some part of the model, and it may be toggled on and off by using the F9 key.

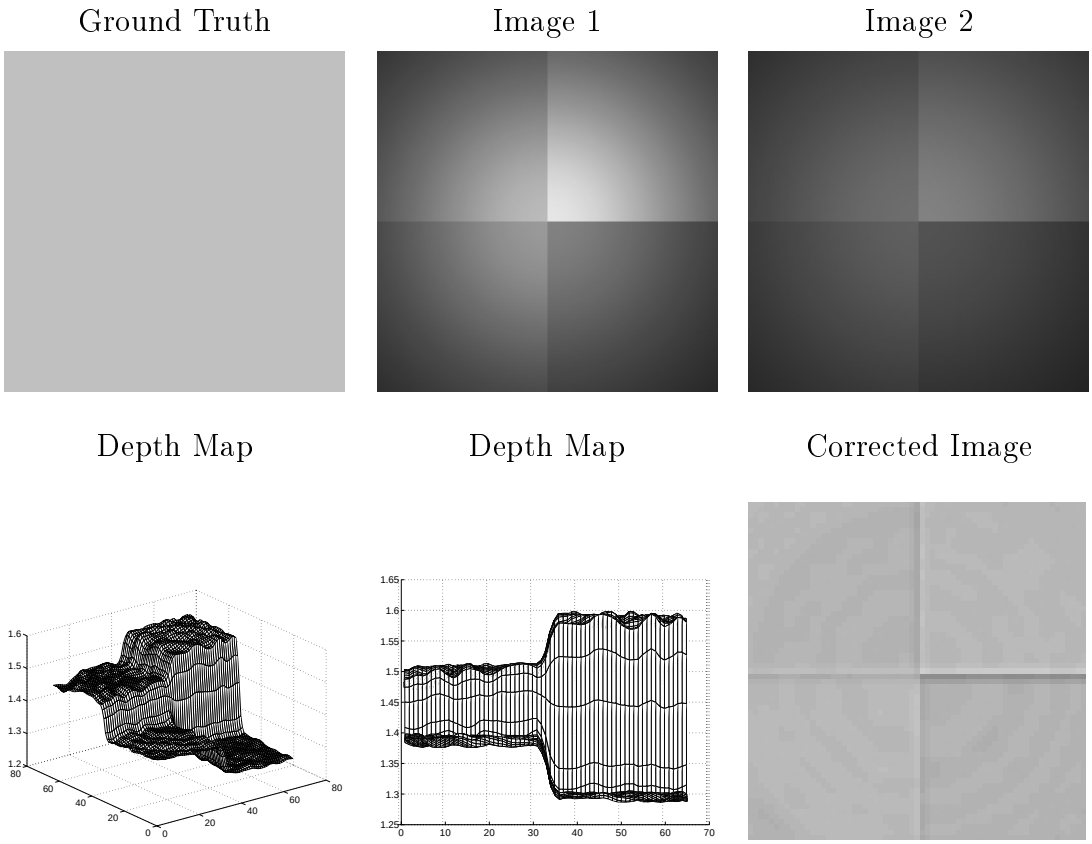

Figure 13: Results of rendering four cubes in the Socrates OpenGL simulator. The top-right cube in the captured images is the closest to the camera at 1.3 units. Moving in a counter-clockwise direction, the cubes are 0.1 units further back– 1.4, 1.5, and 1.6 units respectively. Depth maps show clearly the effectiveness of our depth recovery method, and the corrected image (bottom-right) comes very close to reclaiming the original reflectance function (top-left).

3.3.3 Results

Figure 13 shows the raw data produced by Socrates for the first experiment. Four cubes, with sides of unit length, were positioned in a 2x2 array. For this particular model, the camera was placed at a position 2.0 units from the origin, and facing down the negative z axis. The upper-right cube was positioned the closest to the camera, and each cube was placed incrementally further away from the one clockwise to it. From upper-left to lower-right cube, the cubes were located 1.3, 1.4, 1.5, and 1.6 units from the camera. The face of

each cube had a uniform reflectance function corresponding to the gray value specified by the RGB-tuple (0.75, 0.75, 0.75).

For both images, the light was positioned between the camera and the object, on the z-axis. This was facilitated by OpenGL's ability to "invisibly" place a light within the virtual scene.

The results of feeding these images into the polynomial approximator are shown with the images. These depth estimates represent only the middle quarter of the images captured. OpenGL does not provide any means by which to increase the brightness of an individual light source. Due to this, light attenuated incredibly quickly within the models, and it was difficult to collect data outside of a small range closely proximal to the objects themselves. Moving the light too close caused saturation in the image, while moving the light too far pushed OpenGL past the limits it could reasonably approximate quadratic attenuation with only 8 bits per color per pixel.

Results are on target. An important artifact of the polynomial approximator notable from figure 13 is the steady increase in error with distance. While the cube closest to the light is sharply defined, the one furthest markedly shows a drop in sharpness of the distance estimate. Depth approximations are still well within the useful range though.

To illustrate the effectiveness of the depth estimate, we attempted to recreate the original reflectance function of each cube, with the results shown as the "corrected" image in figure 13. With a perfect depth estimate, this image would be a uniform gray. The most notable exception to this is the line between the nearest and furthest cubes. The furthest cube is also distinctly a different shade of gray from the other three. This is precisely what we would expect as the furthest surface is also the one we have the least information about. Error along the edges of other cubes is minor.

The polynomial approximator is a technique designed to qualitatively improve digital

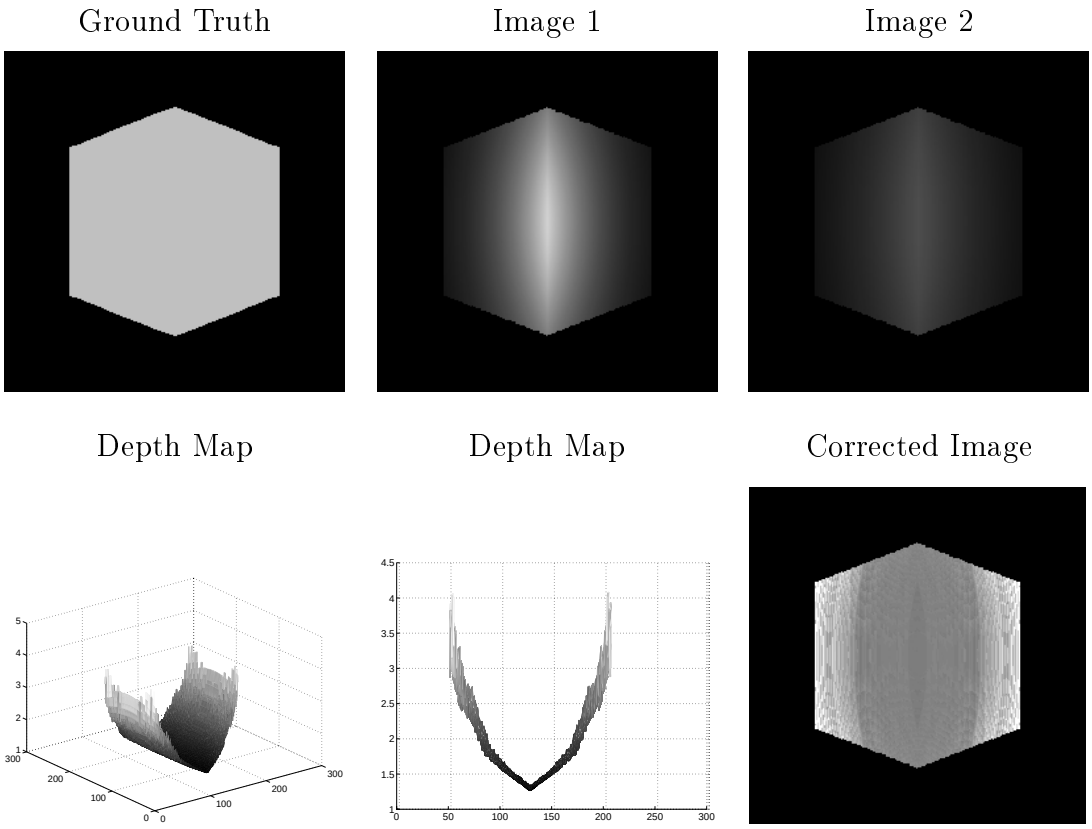

Figure 14: A unit cube, placed at the origin, and rotated 45 degrees about its y-axis. The flashes range has fallen off too quickly to provide good information for depth recovery at the far, visible edges of the cube. As a result, the corrected image shows instabilities, the white saturation, where depth information is unavailable.

image. Each of the models that follow hope to illustrate many of the strong points and a few of the failings inherent to the method.

Figure 14 depicts a unit cube placed at the origin. The cube is rotated 45 degrees about the y-axis. The camera is 2 units away from the origin.

Careful consideration of the depth maps shows that results preserve the angularity of the cube within a good range of its edge. Beyond a certain point, however, the error begins to render the edges non-linear. A brighter flash or more ambient light could help to correct for this effect, and we will consider the latter momentarily.

The corrected image gives a good idea of where the error becomes severe. White regions

represent the edges of the usable depth data. Parts of the cube are thus cut out for lack of information at their depth. This is not reflective of the range of which a consumer-grade flash would be capable, and is in fact some twenty times less. As a result, when taking pictures within the flash's range, one would not expect to encounter this gross a falloff and lack of information about depth.

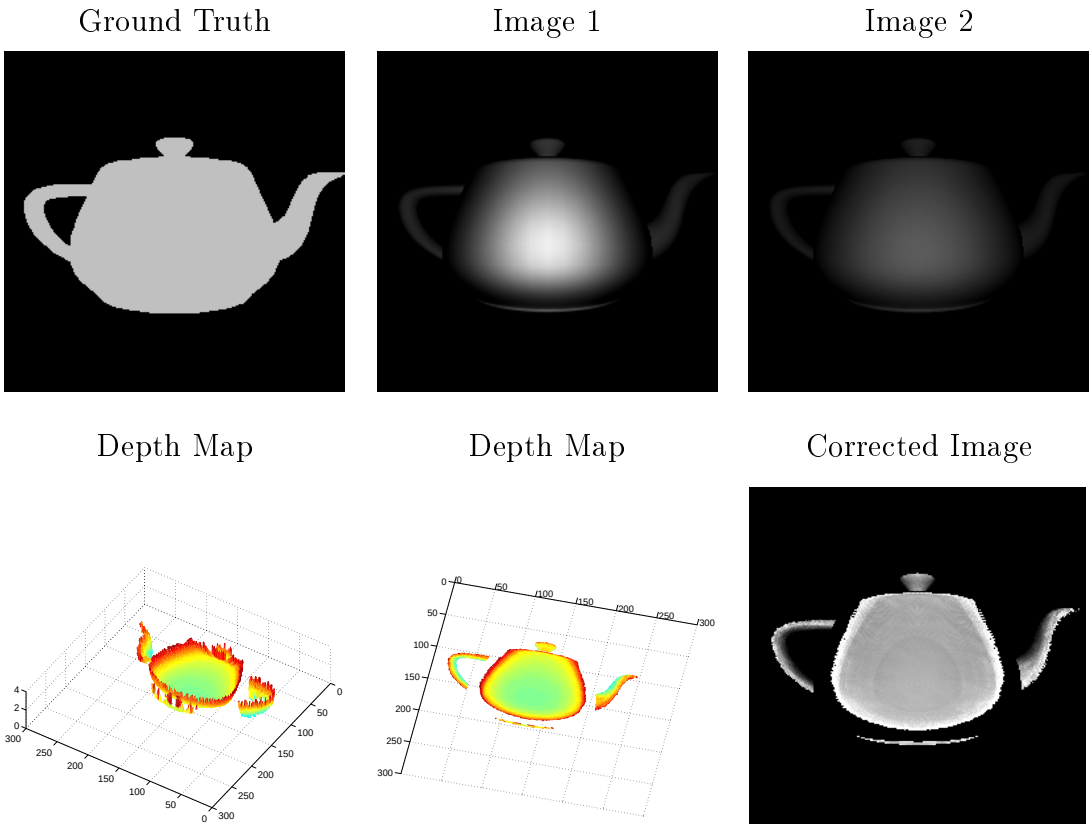

Figure 15: The OpenGL teapot rendered at the origin. As with the rotated cube, the flash's range is too small to illuminate the entire teapot, and depth information is thus unavailable at the teapot's edges.

The quintessential OpenGL model has, of course, been included. Rendered at the origin, with no ambient light, we achieve some satisfying results, but not without room for improvement. As in the case of the rotated cube, the extremely limited range of the OpenGL light source emphasizes the lack of information about the furthest parts of the model. This is best visualized by the corrected image.

A rewarding direction for future research might be to explore how adding ambient light to the scene affects recovering information about the depth map as a whole. As already noted, the ambient term would introduce significant error into the depth map, but most often the error would be of a uniform nature. Areas of an object lit by ambient light, and not affected

by the flash, would cancel to one in the image ratio, as opposed to the current situation where a total lack of information makes the ratio zero– a small amount of information, but conceivably exploitable nonetheless.

4 Conclusions

Overcoming the lack of information about depth for an individual image is one of the most researched problems in image science. Exploiting the quadratic nature of a point light source is a simple and elegant method for recovering some information about depth. Using our technique, we were able to generate excellent depth maps under a variety of different operating conditions– including those most commonly encountered by the amateur photographer.

Understanding the power of the polynomial approximator involves having an appreciation for the huge variety of different image manipulations that can be accomplished with an accurate depth map. As already shown with the OpenGL results, recovering the original reflectance function of the objects in the scene is a trivial operation. With this reflectance function, we are free to create new and accurate images of the same scene under whatever conditions we wish to produce.

Many digital cameras include an LCD display for viewing the image immediately following capture. One compelling idea would be to allow users to literally manipulate the light source in the image after it has already been captured. Perhaps a set of cursors could be built into the back of the camera for translating the light about the scene captured.

More complicated software could readily be made available on home personal computers for adding multiple light sources, changing the characteristics of individual light sources, and even changing the attenuation of light sources.

Although the depth map does not provide information about the "back" of the image subject, enough information is usually available to successfully rotate or skew some part of the subject in the image. Thus, a person could be "nudged" a little to the right. A mug shot could be turned to give more of a profile.

In effect, we have an incredibly inexpensive and fast means for recreating the three-

dimensional structure of an object. Producing accurate three-dimensional models digitally is a hugely researched field, and many of the most popular solutions cost orders of magnitude more than the polynomial approximator.

Manufacturing a device to use the polynomial approximator would involve fabricating a few custom integrated circuits, and building a camera with either two flashes, or some means of quickly refocusing one flash. Most cameras already have dozens of integrated circuits on-board, and could easily accommodate hardware for performing the approximator's trivial set of calculations. A new polymer-based lens that can electronically be refocused is currently in research and development, but actually putting two flashes on opposite ends of a camera is still a reasonable solution. Most professional cameras use separate mounts for the flash, which would make it even easier to build such a solution.

Directions for further research on the topic should include a round of empirical experiments using an actual digital camera or a traditional camera with scanned images. We were not able to run such experiments, but feel confident that the results will be as encouraging as the virtual experiments.

Digital cameras are poised to fundamentally change the way we think about photography. Depth from flash photography is an excellent means of exploiting the advantages inherent to having images already stored in a digital format. It should continue to be explored as a compelling means to improve photography for amateurs and professionals alike.

5 Appendix I

This appendix lists the MATLAB code we used for simulating the polynomial approximator and its sensitivity to some environmental factors. Using this script, one can model the approximator's performance for varying amounts of ambient light, noise, and bit arity.

filename: df.m

```
function [D,IM,IMstd,snr] = df( offset, delta, ambient, noise, quantize )

AMBIENT     = ambient; % additive ambient term (percentage)
NOISE      = noise;    % additive noise (percentage)
QUANTIZE    = quantize; % quantize (num bits); 0 for double precision


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% SYNTHETIC IMAGES

Z     = 20;    % position of plane
LR      = 100;      % light intensity times reflectance
del1    = offset;       % z-axis offset of light #1
del2    = offset + delta;     % z-axis offset of light #2

imdim     = 128;   % image dimensions
f     = 0.5;   % focal length
mmpix     = 0.005; % mm -> pix conversion

[x,y]     = meshgrid( [-imdim/2:imdim/2-1], [-imdim/2:imdim/2-1] ); % image
X     = mmpix*x*Z/f;         % world
Y     = mmpix*y*Z/f;         % world
D0     = sqrt( X.^2 + Y.^2 + Z^2 );   % ground truth

D1    = X.^2 + Y.^2 + (Z+del1)^2;    % diplacement along Z for light #1
D2    = X.^2 + Y.^2 + (Z+del2)^2;    % diplacement along Z for light #2
im1     = LR ./ D1;         % render image #1
im2     = LR ./ D2;         % render image #2

if( AMBIENT )
    a     = AMBIENT/100 * mean( [mean(im1(:)) mean(im2(:))] );
    im1     = im1 + a;
    im2     = im2 + a;
```

```
end
if( NOISE )
    n    = NOISE/100 * mean( [mean(im1(:)), mean(im2(:))] );
    noise1    = n * (rand(size(im1))-0.5);
    noise2    = n * (rand(size(im2))-0.5);
    snr1    = 10 * log10( std(im1) / std(noise1) );
    snr2    = 10 * log10( std(im2) / std(noise2) );
    snr    = (snr1 + snr2)/2;
    im1     = im1 + noise1;
    im2     = im2 + noise2;
else
    snr     = 0;
end

if( QUANTIZE )
    levels = 2^QUANTIZE - 1;
    im1 = round( im1 * levels ) / levels;
    im2 = round( im2 * levels ) / levels;
end

minval     = min( min(im1(:)), min(im2(:)) );
maxval     = max( max(im1(:)), max(im2(:)) );
subplot(3,3,1); imagesc( im1, [minval maxval] );     % image #1
axis image off; colormap gray; title( 'image #1' );
subplot(3,3,2); imagesc( im2, [minval maxval] );     % image #2
axis image off; colormap gray; title( 'image #2' );
subplot(3,3,3); mesh( D0(1:4:imdim,1:4:imdim) );    % distance map
axis( [1 imdim/4 1 imdim/4 min(D0(:)) max(D0(:))] );
axis square;
title( 'distance' );
drawnow;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% QUADRATIC SOLUTION

x2    = (mmpix*x/f).^2;
y2    = (mmpix*y/f).^2;
a    = (im1.*x2 + im1.*y2 + im1) - (im2.*x2 + im2.*y2 + im2);
b    = 2*im1*del1 - 2*im2*del2;
c    = im1*del1^2 - im2*del2^2;
```

```
blur    = [1 4 6 4 1];
blur    = blur / sum(blur);
a    = conv2( conv2( a, blur', 'same' ), blur, 'same' );
b    = conv2( conv2( b, blur', 'same' ), blur, 'same' );
c    = conv2( conv2( c, blur', 'same' ), blur, 'same' );

Zquad    = (-b + sqrt( b.^2 - 4*a.*c )) ./ (2*a);
if( sum(abs(imag(Zquad(:)))) > 1e-3 )
    error( 'imaginary solution' );
    return;
end
X2    = ((mmpix*x/f).*Zquad).^2;
Y2    = ((mmpix*y/f).*Zquad).^2;
Z2    = Zquad.^2;
Dquad    = sqrt( X2 + Y2 + Z2 );

subplot(3,3,6); mesh( Dquad(1:4:imdim,1:4:imdim) ); % ground truth
axis( [1 imdim/4 1 imdim/4 min(D0(:)) max(D0(:))] );
axis square;
title( 'quadratic' );

Dquad1    = X2 + Y2 + (Zquad+del1).^2;     % correction factor #1
Dquad2    = X2 + Y2 + (Zquad+del2).^2;     % correction factor #2
im1q    = im1 .* Dquad1;         % corrected image #1
im2q    = im2 .* Dquad2;         % corrected image #2
imq    = (im1q + im2q)/2;         % average corrected image
subplot(3,3,4); imagesc( im1q, [0 255] ); axis image off; colormap gray;
subplot(3,3,5); imagesc( im2q, [0 255] ); axis image off; colormap gray;
drawnow;

Drms    = sqrt( mean ( (Dquad(:)-D0(:)).^2 ) );    % RMS errors in Z and IM
D = Drms;
imrms    = sqrt( mean ( (imq(:)-LR).^2 ) );
IM = imrms;
imstd    = std(imq(:));
IMstd = imstd;

fprintf( 'errors: [ambient, noise, quantize] = %.2f %.2f %d\n', ...
    AMBIENT, snr, QUANTIZE );
fprintf( 'quad: [D,IM,IMstd] = %.2f %.2f %.2f \n', ...
     Drms, imrms, imstd );
```

References

[1] Dhond, U.R., Aggarwal J.K., *Structure from Stereo - A Review*, IEEE Transactions on Systems, Man, and Cybernetics, vol. 19, no. 6, 1989, pp. 1489-1510.

[2] Farid, H. *Range Estimation by Optical Differentiation*, Ph.D. Dissertation, Dept. of Computer Science, University of Pennsylvania, 1997

[3] B. Jähne. *Digital Image Processing: concepts, algorithms, and scientific applications*, 4th Ed. Springer-Verlag Berlin 1997, pp. 179-180, 195-196.

[4] R.A. Jarvis. *Range from Brightness for Robotic Vision*, 4th International Conference on Robot Vision and Sensory Controls, London, October 1984, proceedings pp. 165-172.

[5] R.A. Jarvis. *Range Sensing for Computer Vision*, *Three-Dimensional Object Recognition Systems*, ed. A.K. Jain, P.J. Flynn, Elsevier, Amsterdam, 1993, pp. 42-43.

[6] Kilgard, M.J. *OpenGL Programming for the X Window System*, Addison-Wesley, 1996.

[7] Sangster, F.I.J., Teer, K. *Bucket-Brigade Electronics*, IEEE Journal Solid-State Circuits, SC-4:131, 1969.

[8] Woo M., Neider J., et al. *OpenGL Programming Guide*, 3rd Ed., Addison-Wesley, 1999.